Simulation of Chemical Kinetics - a Promising Approach to Inference Engines

L. Olac Fuentes and Vladik Kreinovich

University of Texas, El Paso, TX, USA

A rule $A, B \rightarrow C$ from a knowledge base means that if we have reasons to believe in A and B, then our degree of belief in C increases.

Likewise a chemical reaction $A + B \rightarrow C$ means that adding A and B increases the concentration of C. So: Assign to every property A with degree of belief $t(A)$ a fictitious substance with concentration $t(A)$, to every rule - a reaction, simulate the reactions, and here's an inference engine. This idea was tried, but was inefficiently slow.

We used chemical equations for quick reactions instead - and programs worked!

1. FORMULATION OF THE PROBLEM

From the *user's viewpoint* an expert system is a computer-based device that can sometimes substitute an expert: you ask a question and it gives an answer:

question → **BLACKBOX** → *answer*

The *main drawbacks* of existing expert systems from this viewpoint are as follows:

1) sometimes it takes *too much time* to get an answer;

2) sometimes it *doesn't* give any *answer* at all.

From the *designer's viewpoint* an expert system is not a black box: it is a "box" filled with facts and rules plus a mechanism for answering questions that is called an *inference engine*. There exist algorithms like exhaustive backtracking that would finally give an answer to any query for a propositional knowledge base. But the running time for these algorithms grows exponentially with the size of the knowledge base, so we have to delimit these times, and these time limitations explain why sometimes the user gets no answer at all. So from the designer's viewpoint the main drawback of the expert systems is that *inference engines are often too slow*. How to speed them up?

2. CHEMICAL KINETICS APPROACH: MAIN IDEA

How inference engines work

(crude approximation)

In order to figure out how to speed up an inference engine let's describe briefly how it works now. Knowledge in expert systems usually consists of facts $A, B, \ldots$ and *rules* "*if* A, *then* B" or "*if* A, B, *then* C", etc, where $A, B, C, \ldots$ are elementary statements (facts or generalized facts). **Example of a rule:** *if* a person has a fever *and* no rash, *then* it is *probably* the flu. Here A ="a person has a fever", B ="no rash" and C ="flu" are elementary statements (i.e., possible facts). The difference between this rule and the above forms is that it contains the word *probably*. To express such words in expert systems usually numbers from $[0,1]$ are used: 0 means *absolutely false*, 1 means *absolutely true*, $0.5, 0.6, \ldots$ mean that *we are not sure*.

Now we are ready to describe how an inference engine works: if we have a rule "*if* A *and* B, *then* C" in our knowledge base, then if we already have some reasons to believe in A and B, we get some belief in C.

Example. Suppose we have 3 statements A, B, C. **Before** applying any rules we had some reasons to believe in A and B and no prior reasons to believe in C, so our prior beliefs are: $t(A) = 0.6$, $t(B) = 0.4$, $t(C) = 0$. Then we apply the rule $A, B \to C$ and as a result our degree of belief in C increases. E.g., if "and" corresponds to the product of the degrees of belief, then **after** applying the rule we have new degrees of belief: $t(A) = 0.6$, $t(B) = 0.4$, $t(C) = 0.24$.

We placed here this oversimplified description with purpose: it invokes a natural analogy with a well developed area of human knowledge, namely, with chemistry.

Natural analogy with chemistry

Application of a rule $A, B \to C$ is a natural analog of a *chemical reaction* $A + B \to C$ in the following sense: application of the rule increases our degree of belief in C, while when we apply the chemical reaction we increase the *concentration* of a substance C. So:

1) a natural analog of an *elementary statement* A is a *substance* A;

2) a natural analog of a *degree of belief* $t(A)$ is a *concentration* of a substance A;

3) a natural analog of a *rule* $A, B \to C$ is a *chemical reaction* $A + B \to C$.

This analogy can be traced further. E.g., if our belief in A increases, then our belief in the opposite statement "not A" ($\neg A$) decreases and vice versa. Likewise situations exist in chemistry: e.g., adding some acid into the mixture of substances decreases the alkali concentration and vice versa. So a natural analog of A and $\neg A$ are the substances that annihilate each other: $A + \neg A \to$ *media*.

This is not just a vague analogy good for speculations only, it has really been tried as a *heuristic* rule for an inference engine.

How to construct an inference engine : an idea of Matiyasevich

Given: a knowledge base, i.e., a set of statements of the type A; $A \to B$; $A, B \to C$. It is necessary to figure out whether A, B and C are true or not.

We do the following (Matiyasevich (1987)):

1) the formula $A \to B$ means that A implies B. This means also that if B is false, then A is false, i.e., it means also $\neg B \to \neg A$. Likewise $A, B \to C$ means also that $A, \neg C \to \neg B$ and $B, \neg C \to \neg B$. So the first step is that *we write down all these additional equations.* For example, if the initial knowledge base consisted of the statements A; $A \to B$; $A, B \to C$, then after this step the resulting set of statements is: A; $A \to B$; $\neg B \to \neg A$; $A, B \to C$; $A, \neg C \to \neg B$; $B, \neg C \to \neg A$.

2) To each of the elementary formulas $A, B, C, \ldots$ and to their negations $\neg A, \neg B, \neg C, \ldots$ we assign a fictitious chemical "substance" $A, B, C, \ldots, \neg A, \neg B, \neg C, \ldots$ and *write down the correspondent chemical kinetics equation.* In the above example the equations will be: $\to A$, $A \to B$, $\neg B \to \neg A$, $A + B \to C$, $A + \neg C \to \neg B$, $B + \neg C \to \neg A$.

3) *Write down the correspondent equations of chemical kinetics:* (see, e.g., Aris (1969)). Every reaction of the type $A + B \to C$ leads to the terms $+kAB$ in dC/dt and $-kAB$ in dA/dt and dB/dt, where k is some constant. In the above example the resulting equations are:

$$\frac{dA}{dt} = c - kA - kAB - kA\neg C$$

$$dB/dt = kA - kAB - kB\neg C$$

$$dC/dt = kAB$$

$$d\neg A/dt = k\neg B + kB\neg C$$

$$d\neg B/dt = -k\neg B + kA\neg C$$

$$d\neg C/dt = -kA\neg C - kB\neg C.$$

4) *Then we solve this system numerically,* starting from small equal (or random) initial values of all concentrations. In doing this we simulate the interaction of the substances with each other, and we expect that finally we'll get a solution "in vitro": the substances that correspond to *true* elementary statements will stay, and substances that correspond to *false* elementary statements will eventually disappear. If it does not happen for the chosen initial conditions, then we can try again for some other randomly chosen initial concentrations, etc.

Matiyasevich proved that if a system of chemical reactions arrives at a "stable" state and we assign the values "true" and "false" to our elementary statements depending on whether the corresponding substance survived or not, we get the truth values for which all the formulas of the original knowledge base are true.

Remaining problems

Does this system always arrive at a "stable" state? If it does arrive sooner or later, is it *sooner* or *later?* That's the problem we started with: to get an answer we need so much time that it becomes senseless.

3. CRITICISMS AND WAY OUT

Criticisms

Blass and Gurevich (1989) proved that for some reasonable examples the running time of Matiyasevich's algorithm is exponentially big. Actually, they proved that the probability that it reaches a stable state is very low, so we need exponentially many tries with random initial concentrations to reach the stable state after all.

Way out : an idea

Matiyasevich expected that chemical reactions will eventually rule out the unwanted substances, but what Blass and Gurevich actually proved is that the reactions he used are too slow to do that. Why? The idea itself looks promising, so maybe something is wrong with its implementation (namely, with the chemical kinetics equations that we used).

It seems to us that this is really the case. Indeed, where do the chemical kinetics equations come from? Why do we assign to the reaction $A + B \to C$ the equation $dC/dt = kAB$? The reaction occurs

when a (randomly located) molecule of A encounters a randomly located molecule of B. In normal chemical situations concentrations are sufficiently small, so the molecules rarely encounter, and whether a given molecule of A encounters a molecule of B is a random event with the probability proportional to the concentration of B's. These random events for different molecules of A are uncorrelated, therefore the total probability (and hence the total number of interacting pairs) is proportional to the product of the concentrations. So *the very equations of chemical kinetics are based on the assumption that the concentrations are sufficiently small and therefore the reactions are sufficiently slow*, so no wonder that Matiyasevich's algorithm is very slow. Hence to speed it up we must find chemical equations for the case of large concentrations.

For large concentrations molecules do not have to find each other to start the reaction: they are already there, so they start the reaction immediately. Therefore the reaction rate in this case is proportional to the number of matching pairs. E.g., in the reaction $A + B \to C$, if the concentration of A is greater than the concentration of B, then all the available molecules of B will be immediately interacting, and the rate of the reaction will be proportional to the concentration of B. In case the concentration of B is greater, then the rate is proportional to the concentration of A. In both cases it is proportional to the minimum of the concentrations, so the corresponding term in dC/dt will be $\ldots + \min(A, B) + \ldots$

Resulting method : brief description

Given a knowledge base, we:

1) write down all the additional equations with negations;

2) translate them into chemical reactions $A + B \to C$;

3) write the correspondent system of kinetic equations with min instead of a product: $dC/dt = \ldots + k\min(A, B) + \ldots$;

4) solve this systems of differential equations numerically, and assign to the elementary statement A the truth value "true" iff the resulting concentration of A is greater than the resulting concentration of $\neg A$, else assign "false".

(in Section 4 we'll show that this "min" operation really helps: the new algorithm works fine both for the example from Blass and Gurevich (1989) and for random formulas).

Additional ideas. 1) An additional difference between the chemical reactions and the rules is that when we apply a chemical reaction $A + B \to C$, we not only increase the concentration of C, but also decrease the concentrations of A and B. Therefore a chemical reaction leads not only to positive terms $k\min(A, B)$ in dC/dt, but also to negative terms $-k\min(A, B)$ in dA/dt and dB/dt. When we apply a rule $A, B \to C$, then our degree of belief in C increases, but our beliefs in A and B do not decrease. Therefore, when we apply the chemical kinetics equations to knowledge bases, it is reasonable to delete all negative terms and leave only the positive ones.

2) In chemistry we are interested in the dynamics, but here we only want to know who will survive. Therefore, if after some iterations it will be clear, say, that the concentration of the substance A is going to be much bigger than the concentration of $\neg A$, there is no sense to wait until the substance $\neg A$ completely disappears: it is reasonable to "stop" the reactions, delete $\neg A$ and continue the reactions without it.

If the resulting Boolean values do not satisfy the formulas from the original knowledge base, it can mean either that the knowledge base is inconsistent, or that we were too quick to decide that $\neg A$ is going to disappear. In order to check what is the case we can delete A instead and repeat the whole procedure. In other words, if the above-described procedure does not lead to a solution, we can backtrack.

Let's now describe this idea formally.

4. PROPOSED ALGORITHM AND EXPERIMENTAL RESULTS

Preliminary comments

We consider only the propositional case (without variables). This particular case is of certain interest, because it is NP-complete,

and so no algorithm with the running time better than the exponential time is known.

Definitions and formulation of the problem

Suppose that propositional variables $p_1, p_2, ..., p_n$ are given. By a *fact* we mean an expression p_i or $\neg p_i$ for some i. By a *rule* we mean an expression of the type $f_1, ..., f_k \rightarrow f_{k+1}$, where $f_1, ..., f_{k+1}$ are facts. By a *propositional knowledge base* (or simply a *knowledge base*) we mean a finite set of facts and rules.

By an *answer set* we mean a sequence of n truth values $v_1, ..., v_n$. We say that a fact p_i is *true* in this answer set, if $v_i = true$ for that i. We say that a rule $f_1, ..., f_k \rightarrow f_{k+1}$ is *true* if the correspondent propositional formula is true, i.e., either f_{k+1} is true, of one of the facts $f_1, ..., f_k$ is false. We say that an answer set *agrees* with the knowledge base if all its facts and rules are true for this answer set.

The problem is: given a knowledge base, to find an answer set that agrees with it.

Comment. This is, of course, a slight reformulation of the propositional satisfiability problem: by definition an answer set agrees with a knowledge base iff it satisfies a propositional formula, that is a conjunction of all the facts and rules of the knowledge base. Vice versa, any formula in conjunctive normal form (CNF) can be represented this way. But satisfiability problem for formulas in CNF is known to be NP-complete, therefore the problem of finding an answer set that agrees with a given knowledge base is also NP-complete.

Formulation of an algorithm

Step 1. Eliminating facts. First of all, if a knowledge base contains two opposite facts f and $\neg f$, then it is inconsistent and no answer set can agree with it. In case there are no such facts we can eliminate the facts from the knowledge base as follows: Since all the facts f from the knowledge base have to be true, we can assign *true* or *false* to the correspondent v_i and substitute $f = true$ and $\neg f = false$ into all the rules. Namely, if f is one of the premises f_i of the rule $f_1, ..., f_i, ... \rightarrow f_{k+1}$, we can delete f_i from the list of premises (since it's always true); if $\neg f$ is one of the premises, then we can delete the whole rule (because its premises are always false and hence the rule is trivially true). If f is a conclusion of the rule and $f = true$, then again we can delete the rule, since it is trivially true. The only remaining case if when $f = true$ and the conclusion of the rule is $\neg f$. In this case the rule $f_1, ..., f_k \rightarrow \neg f$ means that it is impossible that $f_1, ..., f_k$ are all true and can be therefore rewritten in the form $f_1, ..., f_{k-1} \rightarrow \neg f_k$.

If after this procedure some rules turn into facts, we can eliminate them again, etc. Finally we reduce our problem to the case when a knowledge base contains only rules and no facts.

Step 2. Adding adjacent rules. For every rule $f_1, ..., f_k \rightarrow f_{k+1}$ we add k new rules to our knowledge base:

$$f_1, ..., f_{i-1}, f_{i+1}, ..., f_k, \neg f_{k+1} \rightarrow \neg f_i,$$

where $i = 1, ..., k$. These rules will be called *adjacent*.

Step 3. Iteration process. To every variable p_i and to its negation $\neg p_i$ we put into correspondence a real-valued variable t_i (correspondingly t_{-i}), that will be called our *degree of belief* in p_i (resp. $\neg p_i$). Set the initial values $t_i^{(0)}$ of all these variables to 1. Then apply the following iterative process:

$$t_i^{(k+1)} = t_i^{(k)} + c(min(t_r^{(k)}, ..., t_s^{(k)}) + ...),$$

where c is some positive constant, the sum is taken over all the rules $p_r, ..., p_s \rightarrow p_i$ that contain p_i as a conclusion, and the minimum is taken over the degrees of belief of the premises of that rule.

Comment. This equation corresponds to Euler's integration scheme for a correspondent differential equation, with the values $t_i^{(l)}$ representing the values of the chemical concentrations in the moments $l\delta t$, where δt is an interval between two consequent moments of time. Here $c = k\delta t$, where k is a coefficient of that chemical equation.

Step 3 (continued). The number m of iterations if fixed. After several (m) iterations we

choose a variable p_i, $i = 1,...,n,-1,...,-n$, for which our degree of belief in p_i maximally exceeds our degree of belief in its negation, i.e., for which the ratio t_i/t_{-i} is maximal (if there are several such i, we choose one of them). We assign to this variable the value *true*, i.e., we add p_i as a fact to our knowledge base and go to step 1 again.

When to stop this process and when to backtrack. This procedure has to be repeated until we find the values of all the variables or until we come to a contradiction. If we have found all the values, then we must check whether these values satisfy all the rules of our knowledge base. If it does not satisfy, we must take the last variable p_i whose truth value was decided by an iterative process, and instead of adding p_i add $\neg p_i$ as a fact to our knowledge base. If we have tried both and failed to find a solution in both cases, we must reconsider the previous step, etc.

Example

Let's show how it works on a simple example. This example is here only to clarify the description of the algorithm, not to boast about its efficiency (this will follow later). Let's consider the case when there are two propositional variables a, b and 4 rules $a \to b$; $b \to a$; $\neg a \to b$; $\neg b \to a$. In order to fit our definition of a knowledge base, we must rename the variables into p_1 and p_2. Then the rules from the knowledge base will be: $p_1 \to p_2$; $p_2 \to p_1$; $\neg p_1 \to p_2$; $\neg p_2 \to p_1$. There are no facts, so on the first iteration we skip Step 1. On Step 2 we add rules adjacent to the first two rules of the original database, i.e., $\neg p_2 \to \neg p_1$ and $\neg p_1 \to \neg p_2$ (the third and the fourth rules are adjacent to each other, so we do not have to add anything for them). Let's now follow the iterations of Step 3. We have 4 values t_i here, $i = 1,2,-1,-2$. Their initial values are all 1. The iterative procedure is as follows. For p_1 there are 2 rules with p_1 as a conclusion: $p_2 \to p_1$ and $\neg p_2 \to p_1$. Therefore the sum in the iterative equation for p_1 will consist of 2 terms. The term corresponding to the first rule is just t_2, because there is only one premise in this rule, and the minimum of the one-number set is evidently this very number. Likewise for the second rule the term is t_{-2}, and finally the equations take the form

$$t_1^{(k+1)} = t_1^{(k)} + c(t_2^{(k)} + t_{-2}^{(k)}).$$

Likewise the iterative equations for 3 other variables take the form:

$$t_2^{(k+1)} = t_2^{(k)} + c(t_1^{(k)} + t_{-1}^{(k)}).$$

$$t_{-1}^{(k+1)} = t_{-1}^{(k)} + ct_{-2}^{(k)}.$$

$$t_{-2}^{(k+1)} = t_{-2}^{(k)} + ct_{-1}^{(k)}.$$

The initial value of the vector $(t_1, t_2, t_{-1}, t_{-2})$, consisting of the degrees of belief, is (1,1,1,1). After the first iteration we get $(1+2c, 1+2c, 1+c, 1+c)$, after the second iteration we get $(1+4c+3c^2, 1+4c+3c^2, 1+2c+c^2, 1+2c+c^2)$. Let's choose for simplicity 2 iterations, i.e., $m = 2$.

After the second iteration the values of t_1/t_{-1} and t_2/t_{-2} are equal and greater than 1, and the correspondent values for the negations are less than 1 (and also equal to each other). So according to our algorithm we add either p_1 or p_2 to our knowledge base and go to Step 1 again. In both cases the fact-elimination procedure of Step 1 will lead to $p_1 = p_2 = true$. These values agree with all the rules of the original knowledge base, so we have solved the problem.

Proposed algorithm works fine on the example of Blass and Gurevich

Before applying this algorithm to complicated knowledge bases let's try it on the knowledge bases that were used by Blass and Gurevich (1987) to show that the original Matiyasevich's idea sometimes leads to a too slow algorithm. Their example consists of n propositional variables $p_1,...,p_n$ and n rules $p_i \to p_{i+1}$ for $i = 1,2,...,n-1$ and $p_n \to p_1$. The adjacent rules will be $\neg p_{i+1} \to \neg p_i$ and $\neg p_1 \to \neg p_n$.

If we start with equal initial values for degrees of belief, then, due to the cyclic symmetry of the knowledge base, on every iteration the values of t_i will not depend on i at all. Therefore after m iterations the values of all the ratios t_i/t_{-i} all equal to 1. So we can choose any i, add p_i to our knowledge base, and go to Step 1. If we choose p_i with $i > 0$, then by applying the facts elimination algorithm of Step 1, we get $p_1 = p_2 = ... = p_n = true$. If we choose p_i

with $i < 0$, meaning that we add $\neg p_{|i|}$ to the knowledge base, we likewise come to a conclusion that $p_1 = p_2 = ... = p_n = false$. In both cases the result agrees with the original knowledge base, so we have found a solution (and rather quickly).

If we start with *random* initial values of the degrees of belief, then after m iterations we also choose some p_i, and depending on whether $i > 0$ or $i < 0$ we also get one of the above- described solutions.

Surprisingly this algorithm almost coincides with a neural motivated one

It can be shown (Kreinovich (1987)) that this algorithm coincides with the algorithm, that was proposed by Maslov (1987) as a result of his analysis of human neural networks (the only difference is in our chemical-motivated backtracking). The algorithm of Maslov has been experimentally tried, and on some graph coloring examples worked much better than its competitors (the results are briefly described in Maslov (1987)). Moreover, Maslov tried different functions instead of min and it turned out that experimentally the results with min were the best. Our *chemical interpretation allows us to explain this phenomenon*, because min corresponds to the case of highest concentrations and hence of highest rates.

Algorithm works fine for big random formulas

The choice of the formulas: general idea. We have already mentioned, that a propositional knowledge base is equivalent to the propositional formula in a conjunctive normal form, i.e., is equivalent to a sequence of clauses $f_1 \vee f_2 \vee ... \vee f_k$, where each of f_i is a *literal*, i.e., a propositional variable or its negation. So instead of generating random knowledge bases we can generate random propositional formulas. One of the most natural ways to do it is to fix the number of clauses C, the number of variables V and the number of literals per clause L. In this case we have CL places (L places on each of C clauses) to fill with one of $2V$ possible literals $p_1, ..., p_V, \neg p_1, ..., \neg p_V$. So into every place we put each of the literals with probability $1/(2V)$, and we fill every place independently on how we fill all the others.

The choice of the formulas: what parameters to choose. Depending on the parameters C, L and V, we can either come to a situation where the majority of Boolean vectors satisfy the given formula, or to a situation where such Boolean vectors are quite rare. In the first situation a random choice or a simple backtracking quickly find a satisfying Boolean vector. So in order to make our experiments convincing we consider only the situations with few satisfying vectors, when backtracking is too slow.

It's known how to compute the average number of satisfying Boolean vectors (Purdom & Brown (1987)). The probability that for a given Boolean vector $v_1, v_2, ..., v_n$ a randomly chosen literal is false is $1/2$. But the literals are chosen independently, so the probability that all L literals of the clause are false is equal to $(1/2)^L = 2^{-L}$. A clause is true iff not all its literals are false, so the probability that a clause is true in this very Boolean vector equals to $1 - 2^{-L}$. Likewise independence leads to the fact that the probability that the whole formula is true equals to $(1 - 2^{-L})^C$. The total number N of Boolean vectors in V variables is 2^V, therefore the average number of Boolean vectors satisfying a random formula equals to the product of 2^V and the probability that one vector satisfies it, i.e., to $2^V(1 - 2^{-L})^C$. For our experiment we chose $C = 350$, $V = 20$, $L = 5$; in this case out of a $2^{20} \approx 10^6$ Boolean vectors about 15 satisfy the formula.

"Brute-force" backtracking: running time estimates. The following arguments lead to a crude heuristic estimate of the number of backtracks that we need to find a Boolean vector. Namely, since we consider random formulas, we can consider every backtrack as resulting in one more random Boolean vector to try; the average number of such tries can be estimated as a number T of tries after which the probability to find a vector equals to $1/2$. The probability p to find a Boolean vector after the first try is equal to $(1 - 2^{-L})^C \approx 0.000015$. The probability to miss the solution is therefore $1 - p$, and the probability to miss the solution after T tries equals to $(1 - p)^T$. Therefore T is deter-

lined by the equation $(1-p)^T = 1/2$. So $\approx \ln(1/2)/\ln(1-p) \approx 50,000$.

Our experiments confirmed this crude estimates: actually, in some cases backtracking did not lead to any solution in reasonable time, but the average for the cases when it worked was well above 10^4.

Our algorithm: experimental results. Our algorithm worked in all cases; the average number of backtracks was 57, the greatest was 252: hundreds times smaller than for the usual backtracking.

Experience of actual application :

Diagnosis of logic circuits

We applied our methods to the problem of locating defective gates in logic circuits. We consider the realistic case, when we know that the circuit is defective (because the actual output is different from what we expect), but we have no a priori statistical information (how often are different gates defective). There exist several sets of gates, such that if we assume that all the gates form this set are defective, we explain the observed output. Reiter (1985) proposed to choose a set with minimal number of elements. He also proposed an algorithm that is based on analyzing first sets with one element, then two-gate sets, etc. This algorithm works fine for small circuits, but for actual circuits with many gates it can take a long time.

As an alternative we reformulate diagnosis as a satisfiability problem: we introduce the propositional variables that express the input, output and all the intermediate signals, add a variable for every gate that is true iff this gate is not defective, and express the signal processing in the gate by a corresponding propositional formula. For example, if A represents an AND gate, p and q its inputs and r its output, then the formula is $A \rightarrow (r = p\&q)$. Then we apply the above algorithm to this formula, with the only difference that we take $t_{-i} \ll t_i$ for p_i that represent the gates. This is done because most of the gates are normally good, so for a given gate we have more reasons to believe that it is good than that it is defective.

The resulting algorithm worked quickly both for examples from Reiter (1985) and for the actual defective circuits (for details see Fuentes (1991)).

ACKNOWLEDGEMENTS

This work was supported by NSF Grant CDA-9015006 and NASA Grant NAG 9-482. The authors are greatly thankful to M. Gelfond, Yu. Gurevich, V. Lifschitz and N. B. Maslova for valuable discussions.

REFERENCES

Aris, R. (1969) *Elementary chemical reactor analysis.* Englewood Cliffs: Prentice-Hall.

Blass, A. & Gurevich, Yu. (1989). *On Matiyasevich's nontraditional approach to search problems.* Information Processing Letters, 1989, 32, 41–45.

Fuentes, L.O. (1991) *Applying uncertainty formalisms to well-defined problems.* Unpublished master thesis, University of Texas at El Paso.

Fuentes, L.O. & Kreinovich, V. (1990) *Simulation of chemical kinetics as a promising approach to expert systems.* Proceedings of the Southwestern Conference on Theoretical Chemistry (p. 33), El Paso, TX.

Kreinovich, V. (1987) *Semantics of Maslov's iterative method.* Problems of Cybernetics (Moscow), 131, pp. 30–63 (English translation to appear in Amer. Math. Soc.)

Maslov, S. Yu. (1987) *Theory of deductive systems and its applications.* Cambridge, MA: MIT Press.

Matiyasevich, Yu. (1987). *Possible nontraditional methods of establishing satisfiability of propositional formulas.* Problems of Cybernetics (Moscow), 131, pp. 87–90.

Purdom, P.W. & Brown, C.A. (1987). *Polynomial-average-time satisfiability problems.* Inform. Sci., 41, pp. 23–42.

Reiter, R. (1985) *A theory of diagnosis from first principles.* Technical Report, Computer Science Dept., University of Toronto.